PlusApps: Towards a Privacy Risk Analysis for Android Plus Applications

Abdullah J. Alzahrani
Computer Engineering Department
College of Computer Science and Engineering
University of Ha'il, Hail, Saudi Arabia

***Abstract*—The Android platform leads the mobile operating system marketplace and subsequently has drawn the interest of malware authors and researchers. The significant number of proposed malware detection techniques, classification models and practical reverse engineering solutions are insufficient and there is a lack of perfection. Also, the number of Android apps has increased significantly in recent years, as has the number of apps revealing confidential data. It is essential to investigate the applications and make sure that none of them are leaking privacy data, and consequently a privacy leak analysis approach is needed. Therefore, this paper investigates plus apps behavior and data leakages with a machine-learning algorithm to determine the best features for differentiating plus apps from original apps. The result of the analysis discloses that the SVM classifier presents the greatest accuracy. Further investigation demonstrates that the classifier with the ranking algorithm that uses correlation coefficient (CorEvel) and information gain (InfGain) methods offers more exceptional precision than the other correlation algorithms. The result of this experiment proves that the ranking algorithm is able to decrease the dimension of features and produce an accuracy of 96.60%.**

***Keywords*—Android security; malware detection; permission analysis; privacy risk; plus application**

I. Introduction

Operating systems (OSes) such as Android used in various applications today can be prone to certain risks. Since its inception, Android has come a long way and is prevalently used today. Android has changed the world of smartphones. However, certain risks, such as privacy leakage, can influence the use of Android applications. Android applications can be attacked by malicious codes. According to Zhang et al. [1], malicious code is the "general term used for various hostile or intrusive software, such as viruses, worms, Trojans, spyware, botnets, Rootkits, and backdoors, among others". Malicious code can steal information and essential data from computer users, which greatly affects a user's privacy. When malicious code gets access to a user's personal data and computer, controlling illegal computer systems and cyber source may be possible. In this case, the computer and network credibility, integrity and availability can be destroyed. Meanwhile, Casey [2] noted that malicious code is usually not created using a robust software development lifecycle, with the essential testing and assessment phases needed to work out bugs. Because of this, attackers can crash a target application or OS, which then serves as a warning that a security issue exists. An administrator could not treat every OS or application error as an attack, but certain characteristics that a security issue exists can be looked into. Some of these include crash outcomes after opening an e-mail attachment and after viewing a certain web page in a web browser. There are several diverse Android applications that can be accessed in different parts of the world. But in this variety, there are those that are considered the most popular based on the number of downloads. The top five of the top 20 list of Android applications presented by Price [3] are shown in Table I. A plus application is an APK used to modify the features of the original app for Android. WhatsApp application versions will be investigated, both the plus and the original version from its existence.

TABLE I. The 5 Most Popular Android Apps in the Google Play Store.

No.	Android Application	Description	No. of Downloads
1.	WhatsApp	Instant messaging tool, under the ownership of Facebook since 2014	5.875 billion
2.	Facebook	Popular social networking site	5.478 billion
3.	Facebook Messenger	Instant messaging	3.756 billion
4.	Instagram	Photo and video-sharing social networking service	2.796 billion
5.	Subway Surfers	Game application	1.249 billion

As mentioned earlier, smartphones can be prone to leakage of a user's sensitive and personal data. Data leakage is known to be a serious threat to individuals and enterprise operations, as loss of sensitive information can result in significant reputational damage and financial losses and can be detrimental to the long-term stability of an organization [4]. Apparently, the concept of user awareness in the case of smartphone leakage does not only cover the technical aspects of the device, such as the functioning of Android applications. Alsaleh et al. [5] underscored that both human and technological factors are involved in the multi-dimensional problem of security threats in smartphones. There are also social factors representing the users in terms of user awareness and behavior for securing their smartphone and smartphone applications. For example, Alsaleh et al. [5] found that some smartphone users still chose to share their private data via instant messaging platforms, such as WhatsApp, even though they were aware of the privacy risk. This implies that they accepted the risks associated with their less protected sharing habits as the services offered by these platforms allow them to easily communicate with people they know. This implies that the smartphone's convenience and usefulness, alongside its features and applications, can sometimes affect the decision to behave in a risky manner. There is a lack of user understanding of privacy and security risks linked with installing smartphone applications. In a conducted survey, only 17% of users paid attention to the permissions in their applications, including those that grant

application access to the privacy-sensitive, particularly when an application is being installed. The results showed that only 3% of the survey respondents had a full understanding of the permissions screen [6].

Malicious code can wreak havoc on IoT and mobile devices [7]. The functioning of installed applications in mobile devices will also be negatively affected in the form of data breaches or privacy leakage. For example, data leakage can bring serious threats to organizations. Meanwhile, data leakages are privacy-sensitive data transmitted off the smartphone, through the applications in an unexpected manner. It is therefore important for the malware to be detected during leakage to avoid further or extreme damage. However, it is challenging to detect an application as malware, especially when information leakage occurs [7]. In addition, Cheng et al. [4] stressed that detecting internal data leaks is very challenging as the internal breaches typically involve users having legitimate access to the facilities and data. Doing such actions may also be successful without a trace, as the perpetrators are already knowledgeable about the organization and know how to bypass detection. The prevalent use of the Android OS and the fact that Android applications are often downloaded from third party sources makes it essential to accurately detect those that can be malicious [8]. Additionally, the popularity of Android applications opens the door to several threats and risks from malware applications. According to Singh et al. [9], these simultaneously increasing mobile malware apps can perform malicious activities, such as misusing the private information of users when sending messages and accessing their contacts and other information. Apart from these, confidential information stored in mobile devices can also be illegally exploited. Because of these threats, malware classification and identification becomes a crucial issue. However, permission mechanisms can still be considered great defense mechanisms in ensuring that certain applications cannot harm the user data. Because of this, Singh et al. [9] proposed that malware characterization is determined from the manifest file, allowing the user to enhance the efficiency of Android permissions. In this way, the user will be informed of the risks of Android permissions and applications.

Android's features somehow offer threats and risks to users. According to Fang et al. [10], Android security has been established in a "permission-based mechanism" that restricts the access of third-party Android applications to the critical resources on an Android device. However, the permission-based mechanism has been widely criticized, because of the "coarse-grained control of application permissions" as well as "difficult management of permissions by developers, marketers and even the end users" [10]. Some issues arising in Android security are incompetent administration, coarse granularity of permissions, insufficient permission documentation, incompetent permission administration, permission escalation attack, over-claim permissions and TOCTOU (Time of Check to Time of Use) attacks. Other approaches and initiatives concerning Android permissions were also studied [11].

The purpose of this research is to explore the privacy risks in Android plus applications. To overcome the issues surrounding privacy data leakage, a classification model of plus and original apps needs to be employed. This model examines all permissions used in Android plus apps via ranking algorithms and machine-learning approaches. It obtains features from the apps by analyzing plus and original apps and producing the best feature sets that are employed in the classification models. These models improve assessment of spotting data leakages of plus apps as these apps will be observed corresponding to these features sets. This paper presents an analysis of Android application behavior and data leakages using a variety of classification models. It utilizes feature selection techniques that need to nominate attributes that are engaged to construct the classification model to predict unknown samples. The proposed model examines the apps, differentiating plus apps from original apps, and defines the risk level of the apps using a ranking algorithm that uses correlation coefficient (CorEvel) and information gain (InfGain) methods, offering superior exceptional precision to other correlation algorithms.

The rest of this paper is organized as follows: in Section II, overview of Android platform and security; in Section III, the related work is presented; in Section IV, Android plus apps privacy risk analysis is explained; in Section V, the experiment and result is illustrated; in Section VI, the conclusion and future work are summarized.

II. Overview of Android Platform and Security

The Android Platform is defined as the platform for mobile devices that uses a modified Linux kernel and was introduced by the Open Handset Alliance. Applications running on the Android platform are written in Java programming language. The Java classes are compiled into what is known as "Dalvik Executables" and are operated on the "Dalvik Virtual Machine". Although Android is considered as an open development, it is not open for anyone to contribute, especially when a certain version is under development. All of these are undertaken behind closed doors in the Google office. A developer would need the Android SDK in order to create an application for the platform and this would include tools and APIs. The SDK will also be integrated into the graphical user IDEs (Integrated Development Environments) [12].

Android has developed several security mechanisms. Elovici et al. [13] noted that the Android software stack is established on the "Linux kernel" that is utilized for device drivers, memory management, process management and even networking. This is followed by the next level called the "Android native libraries", where several system components in the upper layers are using the said libraries. The libraries are incorporated into Android applications, which can be made possible through Java native interfaces. This is then followed by the Android "runtime" level, which is composed of the "Dalvik virtual machine and the core libraries" (Elovici et al. [13]). These core libraries are written in Java, while also offering substantial subsets of the Java 5 SE packages and some Android-specific libraries. The "application framework layer" is also fully written in Java and covers the Google tools and propriety tools extensions and services. The phone, web browser and email client, among others, are considered as the topmost application layer. Figure 1 shows the list of the security mechanisms embedded in Android.

Indeed, the security system of Android uses the Linux kernel and offers a set of security measures. It also permits a user-based permissions model, process isolation, secure IPC mechanisms and the ability to eradicate unnecessary or

Mechanism	Description	Security issue
Linux mechanisms		
POSIX users	Each application is associated with a different user ID (or UID).	Prevents one application from disturbing another
File access	The application's directory is only available to the application.	Prevents one application from accessing another's files
Environmental features		
Memory management unit (MMU)	Each process is running in its own address space.	Prevents privilege escalation, information disclosure, and denial of service
Type safety	Type safety enforces variable content to adhere to a specific format, both in compiling time and runtime.	Prevents buffer overflows and stack smashing
Mobile carrier security features	Smart phones use SIM cards to authenticate and authorize user identity.	Prevents phone call theft
Android-specific mechanisms		
Application permissions	Each application declares which permission it requires at install time.	Limits application abilities to perform malicious behavior
Component encapsulation	Each component in an application (such as an activity or service) has a visibility level that regulates access to it from other applications (for example, binding to a service).	Prevents one application from disturbing another, or accessing private components or APIs
Signing applications	The developer signs application `.apk` files, and the package manager verifies them.	Matches and verifies that two applications are from the same source
Dalvik virtual machine	Each application runs in its own virtual machine.	Prevents buffer overflows, remote code execution, and stack smashing

Fig. 1. Security Mechanisms Embedded in Android [13]

possibly insecure parts of the kernel in the operating system. Also, it can go further and render additional efforts in order to avert or prevent multiple system users from accessing and eventually exhausting each other's resources [14]. Fig. 2 shows and summarizes the five key security features of Android that should be carefully studied.

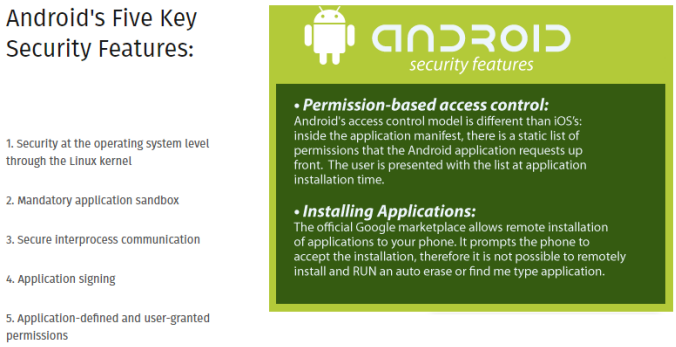

Fig. 2. Five Security Features of Android [14]

Data mining and machine learning are important measures for the security of Android. According to Dua et al. [15], both data mining and machine learning can offer unified reference for a certain machine learning solution towards cyber security issues. It can also supply a foundation for cyber security fundamentals and assess new challenges that detail the cutting-edge machine learning and data mining techniques. It is important for the managers to learn about these while considering the different challenges in data mining and machine learning for security. As Ahmad et al. [16] underscored, a large number of industries are already dependent on network connections, especially those that have sensitive business trading and security matters. In this case, communications and networks are extremely vulnerable to the challenges and threats or risks, such as hacking. Data mining security therefore needs to be applied.

III. Related Work

With the growth in worldwide sales of smartphones, there has been a significant increase in the number of malicious applications that misuse private data without a user's knowledge, publishing it on the online market. Given the massive evolution of the malware, security researchers are required to analyze smartphone applications to identify the intent of the software and to develop defense mechanisms. Previously, application disassemble was achieved by employing tools such as decompilers and runtime debuggers. These techniques require significant amounts of time and are capable of causes errors, depending on the proficiency of the analyst. In the fact an automatic analysis model [17] examines the downloaded applications without human involvement. A key technique in automatic analysis is performing reverse engineering on the application's disassembling, smali code, code decryption, pattern matching, static system call analysis, and Application Programming Interface (API) calls. Meanwhile, malicious code developers are improving their coding skills to find new ways for the malware to evade the detection techniques [18].

Android sensitive data leaks have recently been drawing attention. PlusApps seems to be the first to systematically study a technique to understand users who installed plus apps without knowing its bad side effects. All current Android privacy leakage detection techniques merely identify privacy leakage. Static Taint Analysis (STA) [19] [20] [21] aims to discover the potential sensitive data leaks with the support of deep analysis and program debugging. On the other hand, these methods generally present false positives and are unable to distinguish between user-intended and unintended operations because user intention and context information is absent. On the other side, Dynamic Taint Analysis (DTA) [22] monitors the sensitive data at runtime by using profiling code instrumentation to the original app code. This technique cannot be employed to automatically identify leaking sensitive data in application markets for the reason that privacy leakages are reported while the apps are executed, and dangerous propagation happens.

BLADE [23] identifies malware downloaded from the web by knowing whether it has the user's permission or not. Nevertheless, smartphone applications usually do not require end-user license agreements (EULA) or warnings, even though the user approves the data (e.g. text forwarding). Pegasus [24] spots abnormal behaviors that can be described as APIs and permissions of applications using the historical order and, similar to this paper, it concentrates on identifying malicious application behaviors that are not consistent with the GUI events. However, sensitive data leakages cannot be shaped as app usage of permissions or APIs, therefore several sensitive data leakages cannot be discovered by such techniques. Moreover, Pegasus validates application behaviors based on application-specific properties, which are complicated to indicate with no understanding of application code. Lately, VetDroid [25] improves Dynamic Taint Analysis by creating requirements for sensitive processes. However, the requirement primarily concentrates on the application rationality, not observing each function and the trigger condition for it.

AppIntent [26] investigates user-intended sensitive data transmission on the Android platform. Woodpecker [27] analyzes potential leakages which dissect the reachability of a critical permission from a public, unprotected interface.

Yajin et al. [18] presented inactive content leaks that altered applications to passively reveal application data. However, it does not examine system calls into the Android platform itself. Felt et. al [6] presented privacy and security risk techniques. The researchers conducted an Internet survey on 308 Android users and a laboratory study of 25 Android users. They found that only 17% of users pay attention to the permissions in the applications when an application is installed. Oglaza et al. [11] focused on the Identity Based Access Control (IBAC) models, which were used as permissions-management solutions on mobile devices. The researchers considered the results of a survey from Google in 2013 which showed that French users have on average 32 applications on their Android smartphones. The users would have to manage hundreds of permissions in order to protect their privacy. Apparently, IBAC can be complex aside from its scalability issues.

Some researchers approached data leakage detection by different methods from information flow analysis. Bayes-Droid [28] performs privacy enforcement by examining a comparison between the sensitive data with sinking-values. It can identify sensitive data leakage more precisely than taint analysis techniques, although it only focuses on EIFs and is not applicable to IIF detection. AGRIGENTO [29] utilizes a black-box variance analysis method to show data leakage exposure for obfuscated apps. Given that it only examines sources of sensitive data and network traffic, it cannot maintain locations of IIFs. In addition, it is not an efficient technique for privacy policy enforcement. Barbon et al. [30] uses a hybrid of data flow and quantitative assessment methods to identify IIFs. DAPA [31] also presented a method founded on abstract interpretation framework. Nevertheless, these methods are not relevant to IIFs other than control reliance.

Various researchers have studied permissions-based models, however PlusApp is focused more on the plus apps' behavior and data leakages. The proposed system has the ability to differentiate plus from original apps based on ranking methods and the application's misbehavior in using risky permissions techniques. This model employs Correlation Coefficient (CorEvel) and information gain (InfGain) methods that have the ability to rank the permissions.

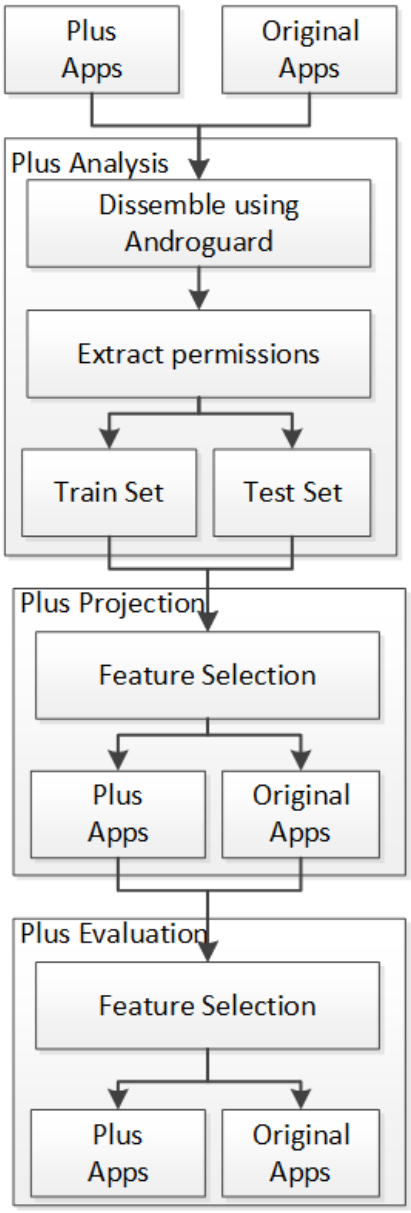

Fig. 3. PlusApps Model Architecture and Components.

IV. Proposed System

The contribution of this research is to examine how permissions are abused by attackers to steal data or damage the mobile device. This paper analyzes these permissions after removing permissions with the normal attribute to see if the proposed system could distinguish between plus and normal apps concerning these permissions. The approach of this research has four phases: plus data collection, plus apps analysis, plus APK projection and plus evaluation. Fig. 3 illustrates the entire workflow of the proposed approach with the corresponding components.

A. Data Collection

In the data collection phase, original and plus app samples are collected that need to be analyzed to be adequate for machine learning approaches. The data creation process began with the data cleansing that was performed to eliminate identical apps and decrease the data amount. Next was the disassembling of the Android applications, making use of reverse engineering tools to retrieve the source code of the apps. Finally the code was analyzed to get the used permissions and the permission occurrences. The first stage, the data cleansing, involves having unique samples and decreasing the features set to accelerate the training phase with highly accurate results. One substantial standpoint that effects data quality is data replication, which impacts the data mining outcomes, and hash techniques were used to identify the identical apps and label these apps by the MD5 hash names. Eliminating replicated data is required in order to have precise and reliable data because inappropriate features have a harmful impact on machine learning [32]. Consequently, replicated samples are excluded from the dataset. There were no duplicate samples found in the original apps, however, many plus samples were duplicated. The dataset was prepared using 454 samples of unique plus

applications. Also, 1,000 samples were downloaded from the Uptodown store. The original samples were separated such that 300 apps were included in the test set and the other 700 APK files assigned to the train set. From the 454 plus APK samples, 317 files were included in the test set and the remaining 137 files assigned to the train set.

B. Plus Apps Analysis

This model involves static analysis for plus application classification that implements machine learning techniques. It analyzes APK applications before executing to determine whether they are harmful applications or not. Many features that can be used towards distinct feature classes are obtained and employed to optimize the feature space. Androguard [33] is utilized for extracting features needed for recognizing plus and original apps. Plus analysis of components consist of two steps: dissembling APK files and feature extraction. An APK file contains all the program's source code, resources, assets, certificates and manifest files. However, the permissions from each APK file are also obtained using the Androguard tool. The distinct features are then used for classification in the first step, followed by the investigation with joint features. The distinct feature model generation requires features that are extracted from distinct categories. Different types of features are used. Permissions: the activities of an Android application must request permission to access sensitive user data. These permissions must be declared statically and a permission-based model is used to offer security for Android architecture. Permission total: this feature set is produced by calculating the set of used permissions by an application.

1) Feature Extraction : A feature in machine learning is a distinct measurable property or characteristic of a phenomenon being observed. This model takes APK files, originally in binary format, as input to the disassembler Androguard tool. One of the output files is the AndroidManifest.xml file that is readable, produced from the input .apk files using Androguard script. The permissions defined in manifest files, which can be extracted, then determines the number of used permissions in each APK file. Also, the amount of permissions used in APK files is a second feature for producing the proposed approach. Since Android APK files and their features have a great importance in Plus Android classification, the basic features are extracted. Moreover, some combined features based on statistical measurements were obtained. The number of dangerous permissions over the total number of permissions requested is an example of the ratio feature.

The total number of permissions of one application and the risk level of each permission are illustrated in Algorithm 1. This algorithm scans all dissembled APK applications and extracts all the features. In addition, it obtains permissions and the occurrence of each permission per application. These permissions are then labeled based on the level of protection and type of risk. These features and their values are saved in a ".csv" file.

C. APK Projection

1) Feature Selection: In this step, feature pruning is conducted to remove the attributes that result in misclassification. After eliminating the inappropriate attributes, joint features to both the modules $(P \cap O)$ are measured. Joint features are given higher priority over the other groups of attributes, such as the union of plus and original features $(P \cup O)$, differentiating original and plus features as they are considered to be irrelevant for the classification of plus samples [34]. Determining the significance of a feature and its attributes is known as feature ranking in machine learning, which has the aim of choosing the most revealing features and refining the performance of learned models [35]. This model utilized correlation Coefficient (CorEvel) and information gain (InfGain) methods

Feature selection is applied to produce the input data into the appropriate size to obtain a subset of k significance features from a set of n features. The nominated attributes are employed to construct the classification model to predict unknown samples. The permissions requested by the applications are classified according to their persistency. These methods are useful to the top 67 common permissions and distinguished 202 features to both plus and original train sets to minimize the feature space.

2) Plus Apps Classifier: From the above phases, the model obtains the requested permissions recognizable with each class label of samples, the total number of the request permissions, and the ratio of risky permissions to the level of risk. To differentiate plus apps from original apps, a classifier is constructed based on selected features that recognize risky applications. The classifier is implemented utilizing four classification algorithms: Support Vector Machine (SVM) [36], Naive Bayes (NB) [36], Decision Tree (J48) [37] and Random Forest (RF) [38], due to their good performance in predicting accuracy.

Support Vector Machine (SVM): SVM tries to find the optimum hyperplane that splits two or more data points from one class on one hand and others on the other hand. In order to separate two classes optimally, the top hyperplane is defined by maximizing margins of both classes. The margins are the distance between the hyperplane and the neighboring point in the classification that can be expressed using the Duality and

Algorithm 1 Total number of permissions of one application and risk level of each permission

Inputs
APKDataset (dissembled APK applications)
Labelled_Risk_Permission

Outputs
FeatureSet (appearance of each permission in Manifest.xml files)

PermiList=[]
$featureSet[] \leftarrow 0$
for e **do**ach Manifest.xml in APKDataset
  $perm \leftarrow 0$
  $currentFeature \leftarrow get\_Read(\text{“}Manfest.xml\text{”})$
  for e **do**ach permission in PermList
    $appearance \leftarrow count\#ofpermissionsin$
    $TypeofRisk \leftarrow compare(permission, Risk_Level)$
    `featureSet[perm] ← featureSet[perm] + appearance`
    $featureSet[perm] \leftarrow featureSet[perm] + perm + TypeofRisk$
    perm++
Return $featureSet[]$

Lagrange Multipliers [39].

Decision Tree (J48): The decision tree can be utilized for unravelling regression and classification issues by learning a training model that that can be used to predict the class with a tree structure [37]. The nodes are the features and leaves that determine class labels. The branches between nodes and leaves are connected with simple decision rules. Predicting a class label for an instance is dependent on the training objects which are all in the root of the tree, and then comparing the root attribute values with the instance's attribute recursively based on selected features. The selection of features is based on an empirical or arithmetic measure. For each recursive stage, the chosen top feature outstandingly decreases the indecision for classification. Thus, the decision tree algorithm naturally holds the function of feature selection.

Random Forest: Random Forest (RF) is an ensemble knowledge method that can be independently learned from a set of decision trees on reduced training sets [38]. To get improved predictive performance, a reduced training set is shaped by arbitrarily sampling with replacement of features. The last choice of classification is completed by choosing between all learned trees. This method performs better than a single tree on classification accuracy.

Naive Bayes (NB) is a supervised learning classifier based on Bayes' theorem that considers the "naive" assumption, finding the relation between every set of elements with equal impact to the target class. The NB classifier considers each feature as unique and does not cooperate with other features. Each class has independent and distinctive features that similarly distributes to the probability of a sample. Naive Bayes is straightforward to develop and is computationally quick, works well on large scale datasets and is not hypersensitive to noise [36].

V. Experiments and Results

This section presents the results of risky permission ranking, application evolution, the extracted explicit privacy leakages features based on level of protection and PlusApps' classification performance evaluation. To further evaluate the performance of the proposed approach, Weka version 3.8.4 software was used [40]. Weka offers resources to train and evaluate classification models for any given features set.

A. Risky Permissions Ranking

To distinguish between plus and original apps, Correlation Coefficient (CorEvel) and information gain (InfGain) methods have the ability to rank the permissions. This ranking uses to determine the plus and original apps and all the data are used to the permission ranking for the experiments. The top 30 risky permissions are illustrated in Table II. As described in the table, the Correlation Coefficient and information gain produced various orders of risky permissions. The results of ranking contain the same risky permissions, with five unique risky permissions in both methods.

Fig. 4 illustrates the existence rate of each top placed permission with CorEvel in original and plus apps. As is shown, the top risky permission differentiate plus apps from the original apps by the rate of occurrence. The top four risky

TABLE II. Top 30 Risky Permissions Ranked by CorEvel and InfGain.

Rank	Correlation Ranker			InfoGain
	Protection Level	Score	CorEvel	
1	Special	0.5854	KILL_BACKGROUND_PROCESSES	KILL_BACKGROUND_PROCESSES
2	Dangerous	0.3549	sticker.READ	sticker.READ
3	Special	0.3549	SYSTEM_ALERT_WINDOW	SYSTEM_ALERT_WINDOW
4	Special	0.3162	WRITE_USE_APP_FEATURE_SURVEY	WRITE_USE_APP_FEATURE_SURVEY
5	Dangerous	0.1696	READ_SETTINGS	READ_SETTINGS
6	Dangerous	0.1696	UPDATE_SHORTCUT	UPDATE_SHORTCUT
7	Signature	0.1584	BROADCAST_BADGE	BROADCAST_BADGE
8	Special	0.1576	WRITE_SETTINGS	WRITE_SETTINGS
9	Dangerous	0.1486	READ_EXTERNAL_STORAGE	BLUETOOTH
10	Dangerous	0.1412	USE_FULL_SCREEN_INTENT	BROADCAST
11	Normal	0.1334	BROADCAST_STICKY	BILLING
12	Dangerous	0.1334	BROADCAST	BROADCAST_STICKY
13	Normal	0.1334	BLUETOOTH	MAPS_RECEIVE
14	Dangerous	0.1334	BILLING	WRITE
15	Signature	0.1301	REQUEST_INSTALL_PACKAGES	CAMERA
16	Signature	0.1294	REGISTRATION	INSTALL_SHORTCUT
17	Dangerous	0.1191	WRITE	READ
18	Normal	0.1191	INSTALL_SHORTCUT	MODIFY_AUDIO_SETTINGS
19	Dangerous	0.1191	READ	READ_GSERVICES
20	Special	0.1191	MODIFY_AUDIO_SETTINGS	CHANGE_WIFI_STATE
21	Signature	0.1191	MAPS_RECEIVE	UNINSTALL_SHORTCUT
22	Dangerous	0.1191	READ_GSERVICES	READ_SYNC_STATS
23	Dangerous	0.1191	CAMERA	READ_SYNC_SETTINGS
24	Special	0.1191	CHANGE_WIFI_STATE	WAKE_LOCK
25	Normal	0.1175	FOREGROUND_SERVICE	WRITE_SYNC_SETTINGS
26	Dangerous	0.103	READ_PHONE_STATE	READ_PROFILE
27	Signature	0.103	UNINSTALL_SHORTCUT	RECEIVE
28	Normal	0.103	READ_SYNC_SETTINGS	VIBRATE
29	Dangerous	0.103	READ_PROFILE	WRITE_CONTACTS
30	Normal	0.103	READ_SYNC_STATS	RECEIVE_BOOT_COMPLETED

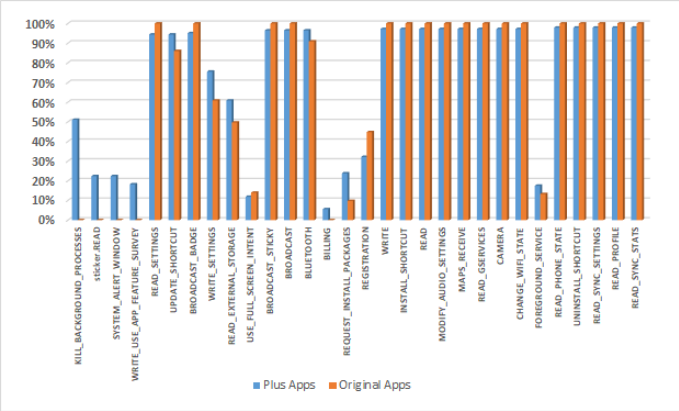

Fig. 4. Occurrence Percentage of the Top 30 Ranked Risky Permissions in Plus Apps and Normal Apps.

permissions are uniquely utilized in the plus apps, ranked by two ranking methods. 40% of the top 30 ranked permissions are dangerous permissions. The occurrence rate for original apps is higher than for plus apps. The top four occurrence rates for plus apps is above 15%, with the following permissions: KILL_BACKGROUND_PROCESSES, sticker.READ, SYSTEM_ALERT_WINDOW, and WRITE_USE_APP_FEATURE_SURVEY, with no occurrence rates for original apps. Most of the ranked permissions are similar due to applying the experiments on the WhatsApp application versions only. The result shows that the use pattern of system modification and data collection permissions is vastly different between the plus apps and the original apps, and many plus apps try to collect data using read and system modification permissions. These permission are consistent with the paper published by Shrivastava et al. [41], which implies that the number of dangerous permissions used to identify the untrustworthy applications and 68% of mobile threats are SMS abuse and data stealer accounts. Billing is also a risky permission that is more likely to be demanded by

plus apps, and CALL_PHONE and INTERNET are sensitive permissions. These latter two permissions are not placed in the top risky permissions demanded by plus and original apps.

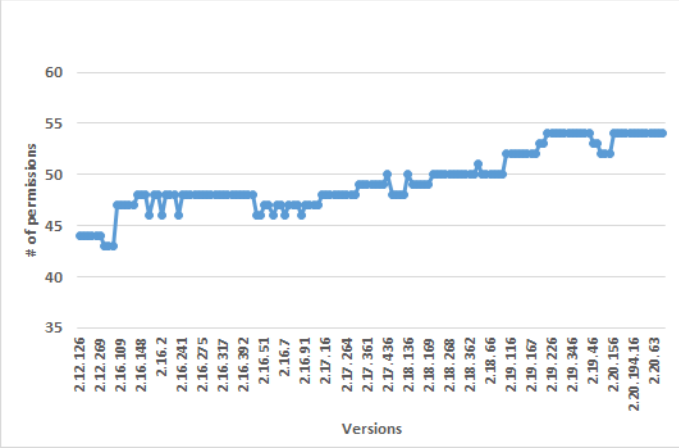

Fig. 5. Permission Evolution of WhatsApps Benign Apps.

Permissions are put into categories of protection levels as “normal”, “dangerous”, “signature” or “special”. The protection level affects whether runtime permission requests are required [42]. The 30 ranking results are almost consistent with the protection levels of Android APIs. The levels of protection are shown in Table II, based on Android 9 API level 28. The number of used permissions is increased in new versions of Android apps, and the occurrence of dangerous permissions in plus apps is significantly higher than the number of dangerous permissions in original apps. There are 6 normal permissions, 13 dangerous permissions, 5 signature permissions and 6 special permissions in ranked permissions. 17 of the top 30 ranked permissions are read, write and change or modification permissions that are categorized as dangerous, signature or special. Most of the plus apps induce users to gain permissions to the app that triggered these permissions to break the system privacy without the awareness of the users. For example, permission KILL_BACKGROUND_PROCESSES is recognized as “special” by Android. However, it is requested by 51% of plus apps in order to allow the plus apps to kill background processes of running applications to activate the suspicious behaviors. This study indicates that most of the ranked permissions are trying to access system resources.

With new development of Android APIs, the requested permissions of Android apps are increased with extra permissions. Fig. 5 presents the permission evolution of WhatsApp’s original apps that shows a significant increase. The permission evolution of WhatsApp’s plus apps are illustrated in Fig. 6, which are similar to the original app’s permission evolution, with the range of 40 to 57 permissions per version.

B. Evaluation Methodology

The evaluation methodology ensures that the PlusApps classifier modules perform well and are able to spot plus apps abnormal behavior intelligently. The development of the classification model consists of four steps: input data selection,

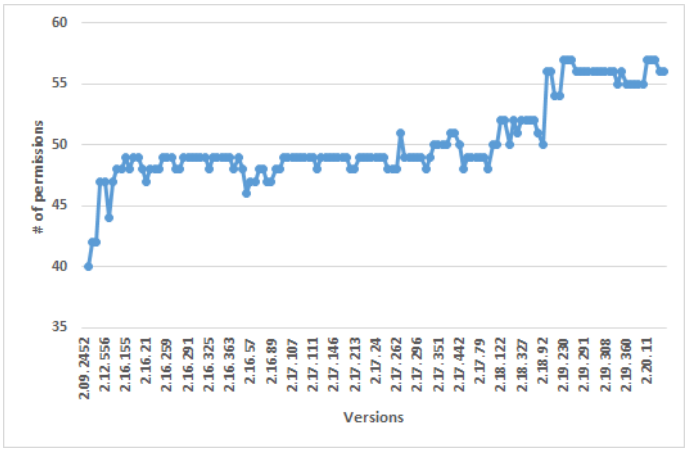

Fig. 6. Permission Evolution of WhatsApp’s Plus Apps.

data pre-processing and splitting (including selecting the features and addressing class), the setting of model parameters and model implementation. The first step in developing the classification models is input variable selection. The APK files vary for each type of dataset that contain plus and original apps. The second step is data pre-processing and splitting. Data pre-processing is an essential step to prepare the data, eliminate outliers and balance the features to the same range. Datasets are typically pre-processed before they can be used for training to speed up convergence. Data is normalized using a linear transformation, which is the method of re-scaling one or more attributes to the range of 0 to 1. Data pre-processing is the process of data conversion that necessitates each data instance is reproduced as a vector of real numbers. Consequently, data has to be converted to numeric data if they are definite attributes [39]. For app classification, they are typically classified to be normal or plus apps, being represented as 0 or 1, before they can be provided to the classifiers. After that the datasets are separated into two parts, which are a training and a testing set. There is no regulation to choose the data partition of training and testing datasets [43]. In many situations, the researchers applied various combinations of data division and adjusted corresponding to the problems.

TABLE III. SUMMARY OF THE 10-FOLD CROSS VALIDATION ACCURACY METRICS.

Accuracy	Kappa	Plus		Normal	
		TP	FN	TN	FP
95.8%	0.902	423	31	970	30

The third step is defining the model parameter and is extremely vital. The appropriate model parameters can enhance the J48, Naive Bayes, Random Tree and SVM classification accuracy performances. There are three types of parameters that should be considered for training the J48 algorithm, which have influence on the resulting decision tree, namely, minimum number of instances in a leaf, use of unpruned trees, confidence factor used in post-pruning and the subtree-raising operation in post-pruning. For the Naive Bayes classifier, all model parameters can be estimated with relative frequencies from the training set namely, class priors, useKernelEstimator

and feature probability distributions. Random Forest is a meta estimator that suits a number of decision tree algorithms on a variety of sub-instances of the data. The most important parameters of Random Forest use averaging to enhance the projecting precision and reduce over-fitting. The sub-instance size is managed with the max_samples parameter if bootstrap=True (default), otherwise the entire dataset is utilized to construct each tree. It then optimizes the random forest, which can be done through a random search using the RandomizedSearch. For the SVM classifier model, there are two important parameters that are considered in the RBF kernel function, namely, regularization C parameter and gamma parameter [44]. The tradeoff cost between minimizing the training error and the complexity of the model are verified by the C parameter, which identifies the non-linear drawing from the low-level space to some high-level dimensional space [44]. In this paper, a parameter search is performed to identify the finest values of parameter C, using trial and error approaches.

The last steps in developing the classification models are model implementation. For the J48 model, the algorithm uses attribute selection to decrease dataset size by eliminating irrelevant/redundant attributes. This algorithm discovers the minimum set of attributes and the resulting probability distribution of data classes, which should be near to the original distribution. In the Naive Bayes classifier, the NB trains until all the features are evaluated, and the one with the highest probability (score) the the winner. For Random Forest, an additional action is to improve the random forest using a random search. Optimization implies obtaining the best hyperparameters for a model on the dataset. The finest hyperparameters differ among datasets and perform model tuning. For SVM classification, the model is trained until the finest sets of parameters (C, Y) are taken. To train and test the generated models, a 10-fold cross-validation technique was performed. In this technique, the instances were divided into training sets (90%) and testing sets (10%), in which the testing set was not part of the training set [45]. Ten unique datasets were produced; in each a different 10% data partition was held out for testing and the rest of the data was used for training. The benefits of k-fold cross validation are that the influence of data dependency is reduced and the consistency of results can be increased. The model's performance was measured using values of true positive rate, false positive rate and AUC (the area under the ROC curve).

C. Performance Evaluation

To differentiate between plus and original application, classification techniques are usually employed in order to evaluate the proposed approaches . The confusion matrix is the best way of representing the classification result (Table V). Due to the two-class nature of the classifier, there are four measures as follows:

- True positive (TP): represents a plus application classified correctly as a plus version.
- False negative (FN): refers to a plus application classified incorrectly as an original version.
- True negative (TN) represents an original application classified correctly as an original version.
- False positive (FP) refers to an original application classified incorrectly as a plus version.

In addition, the performance of different classifier modules is measured using the standard metrics true positive rate, false positive rate and AUC (the area under the ROC curve). The standard metrics extract part of the information from the confusion matrix to produce a numeric value. The higher the true positive rate and the lower the false positive rate, the better the classification is.

1) True Positive Rate (TPR): TPR is the proportion between the plus applications classified correctly as plus version (Equation 1).

$$TPR(recall) = \frac{TP}{TP + FN} \tag{1}$$

2) False Positive Rate (FPR): FPR is the ratio between the number of misclassified original applications and the total number of original applications (Equation 2).

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

3) The ROC curves are used to visualize the relation between true and false positive rates of a certain classifier while tuning it, and on the other hand to compare the accuracy of several classifiers [43]. This metric has two limitations but is nevertheless very effective. The first shortcoming is that the ROC curve is based on the ratio of attack to normal data. It is used for the comparison of detection approaches that run on the same dataset, but the graph of the ROC curve is completely mislead when using it to compare different detection methods that run on different datasets. The second shortcoming is that it may be misleading and basically inadequate for understanding the strengths and weaknesses of a proposed method [43].

Data mining techniques were used to detect the behavior of plus apps based on permissions features. In this experiment, the analysis of different classifiers was used and applied on the dataset to verify whether the app was plus or normal. The results of the analysis were saved in ".csv" file that converted the arff extension in order to process it in Weka. The dataset consisted of 454 plus samples and 1,000 original samples. There were 202 feature vectors with a related label, and the last feature was classification either to plus or normal. The Random Forest classifier with a 10-fold cross validation was used for testing our model. Table III shows the results of evaluation metrics and reveals a high predictive performance.

The precision of recognizing 1,454 different plus and original apps is 95.8% and the Kappa measurement is 0.902, which indicates the performance of the Random Forest classifier using cross validation in this experiment. The False Positive Ratio (FPR) is 3% and 6% for plus and original apps respectively. Moreover, 30 original samples (6.8%) are incorrectly recognized as plus and 93.2% of original apps are correctly identified. 97% of plus apps out of 454 were detected and only 30 plus apps misclassified as legitimate apps. The number of plus apps correctly classified (TP) is 423 and the number of original apps precisely categorized (TN) is 970 apps. Table IV illustrates the results.

TABLE IV. AREA UNDER THE RECEIVER (AUC) AND FALSE POSITIVE RATIO FOR DIFFERENT CLASSIFIERS.

Classifiers	TP Rate	FP Rate	Precision	F-Measure	MCC	ROC Area	PRC Area
SVM	96.60%	5.00%	96.60%	96.60%	92.10%	95.80%	94.90%
NB	93.90%	11.40%	94.10%	93.80%	85.80%	95.50%	96.20%
J48	91.10%	19.30%	92.00%	90.70%	79.50%	85.70%	87.10%
RF	95.80%	5.60%	95.80%	95.80%	90.20%	98.10%	97.90%

TABLE V. THE CONFUSION MATRIX.

	Original	Plus
Original	TN	FP
Plus	FN	TP

1) Discussion and Comparison: Other machine learning algorithms were used with 10-fold cross validation in the experiment. These algorithms were Decision Trees (J48), Naive Bayes, Random Tree and Support Vector Machine (SVM). Table IV shows that all classifiers provided high detection accuracy. An SVM with an RBF kernel classifier accomplished the highest accuracy for the proposed approaches, followed by Random Forest with a high achievement of 95.8% detection rate. The Naive Bayes classifier has a close result with 93.9%, and the Decision Tree (J48) classifier demonstrated the lowest outcome with a 91.10% detection rate. Hence, due to the high similarity of the samples, the Decision Tree (J48) performed the lowest in the proposed model.

Most malware detection approaches consider dangerous permissions are malicious such as SMS permissions. In the paper presented by Wang et al. [21], the SMS-related permissions are always ranked very top risky permissions by the three defined ranking methods. The proposed model has not rated SMS related permission as unsafe permissions in plus application. Also, bill-related and system-related permissions are ranked risky permission in the proposed approach, but the paper published by Wang et al. [21] which not deem these permissions as malicious permissions due to the difference in the behavior of the Plus apps from the malicious application. Other ranked risky permissions are almost similar in both works.

VI. CONCLUSIONS AND FUTURE WORK

The Android platform is an open ecosystem that allow its developers to tailor some of its default features and settings. Many of these settings can be easily customized and use read, write or modified settings of the devices. A plus application is the modified version of an original application with extra features that could result privacy data leakages or misbehavior of these applications. It is crucial that mobile scientific analyzers pay close attention to the types of permissions that these Android applications can request. Therefore, it is important that the many plus applications which are identical to the original application with extra features are investigated in order to study the application's behavior and observe its privacy handling. This paper presents a risk analysis of Android plus applications that investigates the plus apps' behavior and data leakages using classification algorithms. It reveals classification modes to distinguish plus apps from original apps through identifying the effectiveness of Android permissions. The research has shown that it is possible to retrieve some artifacts from plus applications when users installed and used these types of plus applications. These artifacts included kill processes, sticker read, system alert and billing features. The research has also shown that some unique permissions can be used by plus applications that are considered risky features, which results in permissions abuse and data stealer accounts. The analysis reveals that the SVM classifier presented the highest accuracy and additional investigation demonstrates that the classifier with the ranking algorithm use Correlation Coefficient (CorEvel) and Information Gain (InfGain) methods.

REFERENCES

[1] B. Y. Zhang, X. A. Yan, and D. Q. Tang, "Survey on malicious code intelligent detection techniques," *Journal of Physics: Conference Series*, vol. 1087, p. 062, sep 2018.

[2] E. Casey, C. Daywalt, and A. Johnston, "Intrusion investigation," in *Handbook of Digital Forensics and Investigation*. San Diego: Academic Press, 2010, pp. 135–206.

[3] D. Price, "The 20 most popular android apps in the google play store." New York ,USA, 03 2020.

[4] L. Cheng, F. Liu, and D. D. Yao, "Enterprise data breach: causes, challenges, prevention, and future directions," *WIREs Data Mining and Knowledge Discovery*, vol. 7, no. 5, pp. 1–11, 2017.

[5] M. Alsaleh, N. Alomar, and A. Alarifi, "Smartphone users: Understanding how security mechanisms are perceived and new persuasive methods," *PLOS ONE*, vol. 12, 03 2017.

[6] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*, ser. SOUPS '12. New York, NY, USA: Association for Computing Machinery, 2012.

[7] Y. Kim, T. Oh, and J. Kim, "Analyzing user awareness of privacy data leak in mobile applications," *Mobile Information Systems*, vol. 2015, pp. 1–12, 12 2015.

[8] S. Rai, R. Dhanesha, S. Nahata, and B. Menezes, *Malicious Application Detection on Android Smartphones with Enhanced Static-Dynamic Analysis*, 01 2017, pp. 194–208.

[9] P. Singh, P. Tiwari, and S. Singh, "Analysis of malicious behavior of android apps," *Procedia Computer Science*, vol. 79, pp. 215–220, 2016, proceedings of International Conference on Communication, Computing and Virtualization (ICCCV) 2016.

[10] Z. Fang, W. Han and Y. Li, "Permission based android security: Issues and countermeasures," *Computers & Security*, vol. 43, pp. 205–218, 2014.

[11] A. Oglaza, R. Laborde, P. Zarate, A. Benzekri and F. Barrere, "A new approach for managing android permissions: learning users' preferences," *EURASIP Journal on Information Security*, vol. 2017, 07 2017.

[12] Technopedia, "Android platform." New York ,USA, 08 2011.

[13] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google android: A comprehensive security assessment," *IEEE Security Privacy*, vol. 8, no. 2, pp. 35–44, 2010.

[14] V. Code, "Android security: Guide to android os." Burlington, MA ,USA, 01 2020.

[15] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*, 1st ed. USA: Auerbach Publications, 2011.

[16] B. Ahmad, J. Wan, and Z. A. Ali, "Role of machine learning and data mining in internet security: Standing state with future directions," *Journal Comp. Netw. and Communic.*, vol. 18, p. 10, 2018.

[17] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," vol. 44, no. 2, 2008.

[18] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 95–109.

[19] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *SIGPLAN Not.*, vol. 49, no. 6, pp. 259—269, Jun. 2014.

[20] M. Gordon, K. deokhwan, J. Perkins, L. Gilham, N. Nguyen, and M. Rinard, "Information-flow analysis of android applications in droidsafe," 01 2015.

[21] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1869–1882, 2014.

[22] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. USA: USENIX Association, 2010, pp. 393—407.

[23] L. Lu, V. Yegneswaran, P. Porras, and W. Lee, "Blade: An attack-agnostic approach for preventing drive-by malware infections," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2010, pp. 440—450.

[24] K. Z. Chen, N. Johnson, S. Dai, K. Macnamara, T. Magrino, E. Wu, M. Rinard, and D. Song, "Contextual policy enforcement in android applications with permission event graphs," 2013.

[25] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang, "Vetting undesirable behaviors in android apps with permission use analysis," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. New York, NY, USA: Association for Computing Machinery, 2013, pp. 611—622.

[26] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: Analyzing sensitive data transmission in android for privacy leakage detection," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 1043—1054.

[27] M. C. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Systematic detection of capability leaks in stock android smartphones." in *NDSS*, 2012.

[28] O. Tripp and J. Rubin, "A bayesian approach to privacy enforcement in smartphones," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC'14. USA: USENIX Association, 2014, pp. 175—190.

[29] A. Continella, Y. Fratantonio, M. Lindorfer, A. Puccetti, A. Zand, C. Kruegel, and G. Vigna, "Obfuscation-resilient privacy leak detection for mobile apps through differential analysis," 01 2017.

[30] G. Barbon, A. Cortesi, P. Ferrara, M. Pistoia, and O. Tripp, "Privacy analysis of android apps: Implicit flows and quantitative analysis," in *Computer Information Systems and Industrial Management*. Cham: Springer International Publishing, 2015, pp. 3–23.

[31] G. Barbon, A. Cortesi, P. Ferrara, and E. Steffinlongo, "Dapa: Degradation-aware privacy analysis of android apps," in *STM*, 2016.

[32] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *IEEE Data Engineering Bulletin*, vol. 23, p. 20, 2000.

[33] *Androguard*, available at: http://github.com/androguard/androguard./ accessed on Feb 14, 2020.

[34] A. M. Aswini and P. Vinod, "Droid permission miner: Mining prominent permissions for android malware analysis," in *The Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2014)*, vol. 5, no. 1, 2014, pp. 81–86.

[35] I. Guyon, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[36] S. Misra and H. Li, *Noninvasive fracture characterization based on the classification of sonic wave travel times*, 2020, pp. 243–287.

[37] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81—106, 1986.

[38] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5—32, 2001.

[39] C. Burges, "A tutorial on support vector machines for pattern recognition." *Data Mining and Knowledge Discovery*, vol. 2, no. 1, p. 121-167, 1998.

[40] E. Frank, M. Hall, G. Holmes, R. Kirkby, B. Pfahringer, I. Witten, and L. Trigg, *Weka - A Machine Learning Workbench for Data Mining*, 07 2010, pp. 1269–1277.

[41] G. Shrivastava and P. Kumar, "Intent and permission modeling for privacy leakage detection in android," *Energy Systems*, 10 2019.

[42] G. Developer, "Permissions overview," New York ,USA, 07 2020.

[43] S. Alsoghyer and I. Almomani, "Ransomware detection system for android applications," *Electronics*, vol. 8, p. 868, 08 2019.

[44] H.-L. Chen, B. Yang, J. Liu, and D.-Y. Liu, "A support vector machine classifier with rough set-based feature selection for breast cancer diagnosis," *Expert Systems with Applications*, vol. 38, no. 7, pp. 9014–9022, 2011.

[45] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143.